

# AULA 19

# Knuth-Morris-Pratt

CLRS 32.4

# KMP e algoritmo do autômato

O algoritmo **KMP-MATCHER** pode ser visto como uma implementação do algoritmo

**FINITE-AUTOMATON-MATCHER**  $(P, m, T, n)$

```
1   $q \leftarrow 0$ 
2  para  $i \leftarrow 1$  até  $n$  faça
3       $q \leftarrow$  maior  $k$  tal que  $P[1..k]$  é sufixo de  $T[1..i]$ 
4      se  $q = m$ 
5          então “ $P$  ocorre com deslocamento  $i - m$ ”
```

As iterações do algoritmo **KMP-MATCHER** em que o valor de  $i$  **não** é incrementado estão determinando o

“**maior**  $k$  tal que  $P[1..k]$  é **sufixo** de  $T[1..i]$ ”

# KMP e algoritmo do autômato

O algoritmo **KMP-MATCHER** pode ser visto como uma implementação do algoritmo

**FINITE-AUTOMATON-MATCHER**  $(P, m, T, n)$

```
1   $q \leftarrow 0$ 
2  para  $i \leftarrow 1$  até  $n$  faça
3       $q \leftarrow$  maior  $k$  tal que  $P[1..k]$  é sufixo de  $T[1..i]$ 
4      se  $q = m$ 
5          então “ $P$  ocorre com deslocamento  $i - m$ ”
```

O valor de  $i$  só é **incrementado** após a determinação desse

“**maior**  $k$  tal que  $P[1..k]$  é **sufixo** de  $T[1..i]$ ”

Para isto o algoritmo faz uso de uma certa função  $\pi$ .

# Função prefixo

$\pi[q] :=$  maior comprimento de um **prefixo próprio**  
 de  $P[1..q]$  que é **sufixo** de  $P[1..q]$   
 $:= \max\{k : k < q \text{ e } P[1..k] \text{ é sufixo de } P[1..q]\}$

$q$	1	2	3	4	5	6	7	8	9	10	11
$P$	a	b	a	b	b	a	b	a	b	b	a
$\pi$	0	0	1	2	0	1	2	3	4	5	6

$q$	1	2	3	4	5	6	7	8	9	10
$P$	a	b	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	5	6	0	1

# Simulação

$P = a b a b b a b a b b a$

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

a b a a b a b a b b a b a b a b b a b a b b a *T*

1 a b a b

4 a b

4 a b a b b

8 a b a b b a b a b b

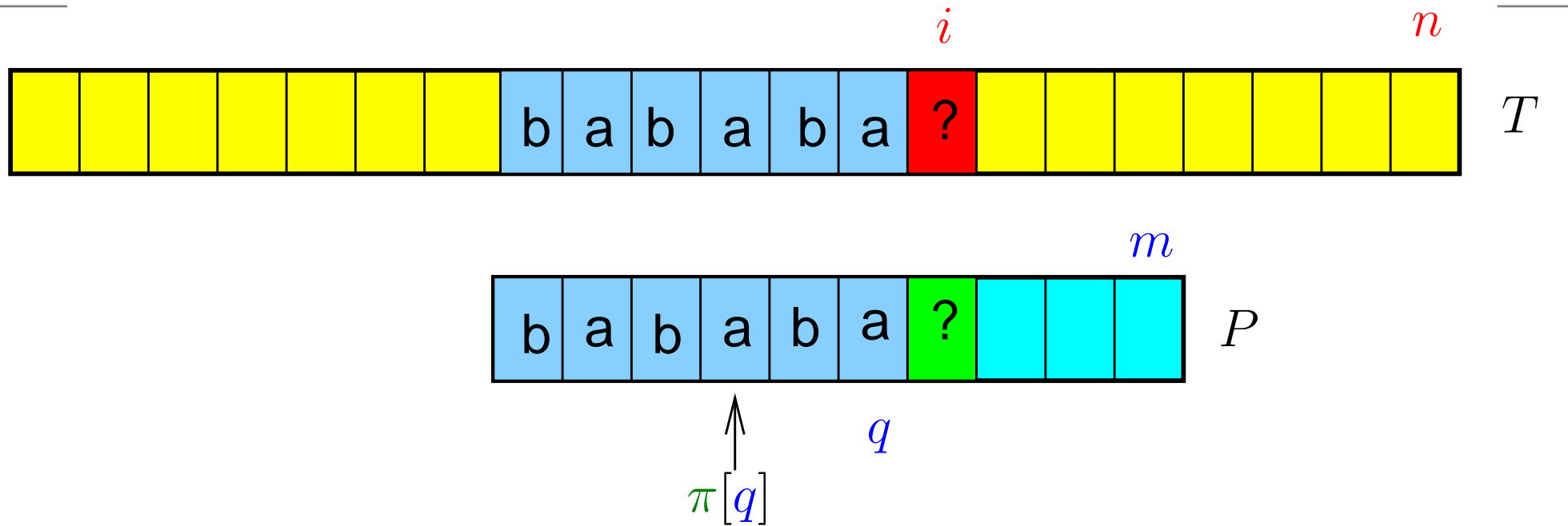
15 a b a b b

15 a b a b b a b a b b a

*i*

1 1 1 3 1 1 1 2 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1

# Iteração



Relações invariantes:

(i0)  $P[1..q]$  é sufixo de  $T[1..i-1]$

(i1)  $P[1..k]$  não é sufixo de  $T[1..i]$ , para  $k \geq q + 2$ .

# KMP-Matcher

KMP-MATCHER ( $P, m, T, n$ )

```
0   $\pi \leftarrow \text{COMPUTE-PREFIX-FUNCTION}(P, m)$ 
1   $i \leftarrow 1 \quad q \leftarrow 0$ 
2  enquanto  $i \leq n + 1$  faça
3      se  $q = m$  então  $\triangleright$  caso 1
4          “ $P$  ocorre com deslocamento  $i - m$ ”
5           $q \leftarrow \pi[q]$ 
6      senão se  $P[q+1] = T[i]$  então  $\triangleright$  caso 2
7           $q \leftarrow q + 1$ 
8           $i \leftarrow i + 1$ 
9      senão se  $P[q+1] \neq T[i]$  e  $q > 0$  então  $\triangleright$  caso 3
10          $q \leftarrow \pi[q]$ 
11     senão se  $P[q+1] \neq T[i]$  e  $q = 0$  então  $\triangleright$  caso 4
12          $i \leftarrow i + 1$ 
```

Suponha  $T[n + 1] = \text{“s}{\text{ímbolo que não ocorre em }} P[1..m]\text{”}$



# Correção

**Relação invariante:** na linha 2 vale que

(i0)  $P[1 \dots q]$  é **sufixo** de  $T[1 \dots i-1]$

Na linha 2 vale ainda que

(i1)  $P[1 \dots k]$  **não** é sufixo de  $T[1 \dots i]$ , para  $k \geq q + 2$ .

Invariante (i1) implica que  $i - m, i - m + 1, \dots, i - q - 1$  **não** são **deslocamentos válidos**, ou seja

$P[1 \dots m]$  **não** é sufixo de  $T[1 \dots i + 1]$

$P[1 \dots m]$  **não** é sufixo de  $T[1 \dots i + 2]$

...

$P[1 \dots m]$  **não** é sufixo de  $T[1 \dots i + m - q - 1]$ .

# Consumo de tempo

Exceto a linha 0, cada linha do algoritmo consome tempo  $\Theta(1)$ .

Caso 2 e caso 4 ocorrem  $n + 1$  vezes (total).

Para cada valor de  $i$ , número de ocorrências do caso 1 e caso 3 é  $\leq m$ .

Logo, o número total de iterações das linhas 2–12  
é  $\leq n + 1 + (n + 1)m = (n + 1)(m + 1)$ .

Portanto, o consumo de tempo total das linhas 1–12  
é  $O(nm)$ .

# Consumo de tempo

Exceto a linha 0, cada linha do algoritmo consome tempo  $\Theta(1)$ .

Caso 2 e caso 4 ocorrem  $n + 1$  vezes (total).

Para cada valor de  $i$ , número de ocorrências do caso 1 e caso 3 é  $\leq m$ .

Logo, o número total de iterações das linhas 2–12 é  $\leq n + 1 + (n + 1)m = (n + 1)(m + 1)$ .

Portanto, o consumo de tempo total das linhas 1–12 é  $O(nm)$ .

**EXAGERO!**

# Consumo de tempo

Exceto a linha 0, cada linha do algoritmo consome tempo  $\Theta(1)$ .

Caso 2 e caso 4 ocorrem  $n + 1$  vezes (total).

O número de ocorrências do caso 1 e caso 3 é  $\leq$  número de ocorrências do caso 2, pois

- $q$  nunca é negativo;
- $q$  é incrementado no caso 2;
- $q$  é decrementado no caso 1 e no caso 3.

Logo, o número total de iterações é  $\leq n + 1 + n + 1 = 2n + 2$ .

Portanto, o consumo de tempo das linhas 1–12 é  $\Theta(n)$ .

# KMP-Matcher

KMP-MATCHER ( $P, m, T, n$ )

```
0   $\pi \leftarrow$  COMPUTE-PREFIX-FUNCTION( $P, m$ )
1   $i \leftarrow 1$     $q \leftarrow 0$ 
2  enquanto  $i \leq n$  faça
3      enquanto  $q > 0$  e  $P[q+1] \neq T[i]$  faça
4           $q \leftarrow \pi[q]$ 
5          se  $P[q+1] = T[i]$  então
6               $q \leftarrow q + 1$ 
7          se  $q = m$  então
8              “ $P$  ocorre com deslocamento  $i - m$ ”
9               $q \leftarrow \pi[q]$ 
```

# Compute-Prefix-Function (grosseiro)

COMPUTE-PREFIX-FUNCTION ( $P, m$ )

```
0   $\pi[1] \leftarrow 0$ 
1   $q \leftarrow 2$ 
2  enquanto  $q \leq m$  faça
3       $k \leftarrow q - 1$ 
4      repita
5           $i \leftarrow k$ 
6           $j \leftarrow q$ 
7          enquanto  $i > 0$  e  $P[i] = P[j]$  faça
8               $i \leftarrow i - 1$      $j \leftarrow j - 1$ 
9              se  $i > 0$ 
10                 então  $k \leftarrow k - 1$ 
11         até que  $i = 0$ 
12      $\pi[q] \leftarrow k$ 
13 devolva  $\pi$ 
```

# Consumo de tempo

linha    consumo de **todas** as execuções da linha

---

0-1	$\Theta(1)$
2	$\Theta(m)$
3	$\Theta(m - 1)$
4-6	$O(m^2)$
7-8	$O(m^3)$
9-11	$O(m^2)$
12	$\Theta(m - 1)$
13	$O(m)$

---

**total**     $\Theta(3m - 1) + O(m^3 + 2m^2 + m)$   
               $= O(m^3)$

# Função prefixo e algoritmo do autômato

O algoritmo **COMPUTE-PREFIX-FUNCTION** pode ser semelhante ao algoritmo

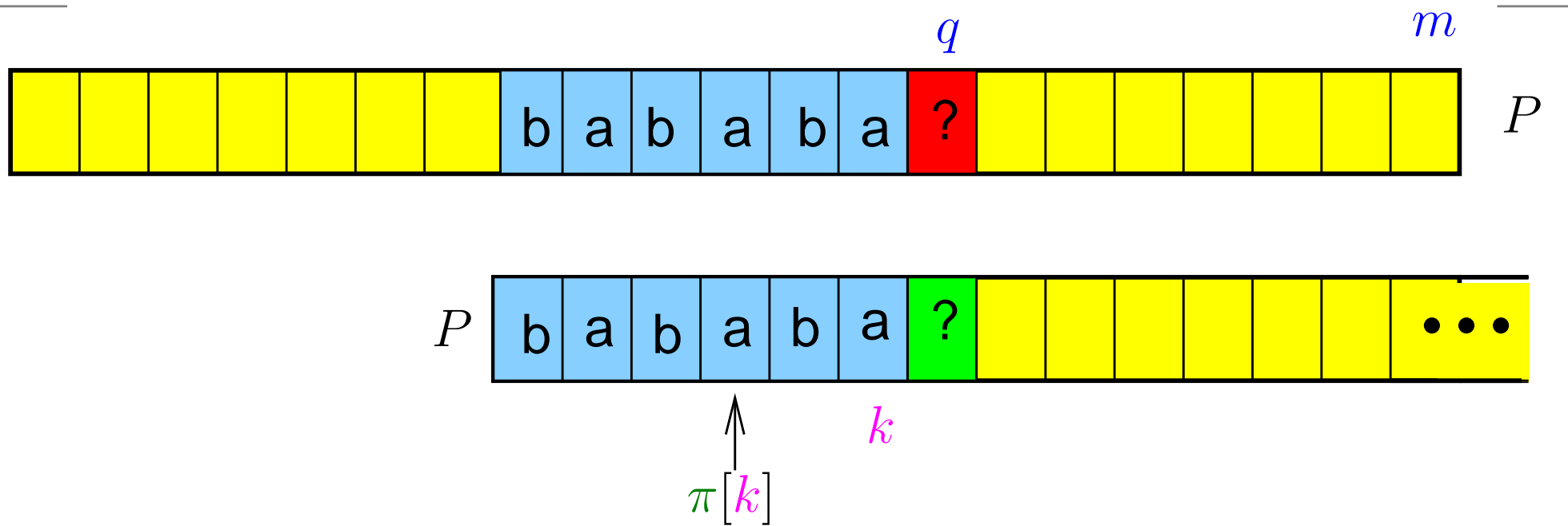
**COMPUTE-PREFIX-FUNCTION** ( $P, m$ )

```
1  para  $q \leftarrow 1$  até  $m$  faça
2       $\pi[q] \leftarrow$  maior  $k$  tal que  $P[1..k]$  é sufixo próprio
2      de  $P[1..q]$ 
```

O valor de  $q$  só é incrementado após a determinação desse  
“maior  $k$  tal que  $P[1..k]$  é sufixo próprio de  $P[1..i]$ ”



# Iteração



Para determinar o

“maior  $k$  tal que  $P[1..k]$  é **sufixo próprio** de  $P[1..i]$ ”  
podemos utilizar os valores de  $\pi$  já calculados.

# Subestrutura ótima

Suponha que  $P[1 \dots k + 1]$  é o maior **prefixo próprio** de  $P[1 \dots q]$  que é sufixo de  $P[1 \dots q]$  ( $\pi[q] = k + 1$ ).

# Subestrutura ótima

Suponha que  $P[1..k+1]$  é o maior **prefixo próprio** de  $P[1..q]$  que é sufixo de  $P[1..q]$  ( $\pi[q] = k+1$ ).

Portanto,

- $P[k+1] = P[q]$  e
- $P[1..k]$  é **prefixo próprio** e sufixo de  $P[1..q-1]$ .

Logo,  $P[1..k]$  é prefixo e sufixo de  $P[1..\pi[q]-1]$ .

# Subestrutura ótima

Suponha que  $P[1..k+1]$  é o maior **prefixo próprio** de  $P[1..q]$  que é sufixo de  $P[1..q]$  ( $\pi[q] = k+1$ ).

Portanto,

- $P[k+1] = P[q]$  e
- $P[1..k]$  é **prefixo próprio** e sufixo de  $P[1..q-1]$ .

Logo,  $P[1..k]$  é prefixo e sufixo de  $P[1..\pi[q-1]]$ .

Se  $k \neq \pi[q-1]$ , então

# Subestrutura ótima

Suponha que  $P[1..k+1]$  é o maior **prefixo próprio** de  $P[1..q]$  que é sufixo de  $P[1..q]$  ( $\pi[q] = k+1$ ).

Portanto,

- $P[k+1] = P[q]$  e

- $P[1..k]$  é **prefixo próprio** e sufixo de  $P[1..q-1]$ .

Logo,  $P[1..k]$  é prefixo e sufixo de  $P[1..\pi[q-1]]$ .

Se  $k \neq \pi[q-1]$ , então

$P[1..k]$  é **prefixo próprio** e sufixo de  $P[1..\pi[q-1]]$ .

Logo,  $P[1..k]$  é prefixo e sufixo de

$$P[1..\pi[\pi[q-1]]] = P[1..\pi^2[q-1]].$$

# Subestrutura ótima (cont.)

Se  $k \neq \pi^2[q - 1]$ , então

# Subestrutura ótima (cont.)

Se  $k \neq \pi^2[q - 1]$ , então

$P[1 \dots k]$  é **prefixo próprio** e sufixo de  $P[1 \dots \pi^2[q - 1]]$ .

Logo,  $P[1 \dots k]$  é sufixo de

$$P[1 \dots \pi[\pi[\pi[q - 1]]]] = P[1 \dots \pi^3[q - 1]].$$

# Subestrutura ótima (cont.)

Se  $k \neq \pi^2[q - 1]$ , então

$P[1 \dots k]$  é **prefixo próprio** e sufixo de  $P[1 \dots \pi^2[q - 1]]$ .

Logo,  $P[1 \dots k]$  é sufixo de

$$P[1 \dots \pi[\pi[\pi[q - 1]]]] = P[1 \dots \pi^3[q - 1]].$$

Se  $k \neq \pi^3[q - 1]$ , então

$P[1 \dots k - 1]$  é **prefixo próprio** e sufixo  
de  $P[1 \dots \pi^3[q - 1]]$ .

Logo,  $P[1 \dots k]$  é prefixo e sufixo de  $P[1 \dots \pi^4[q - 1]]$ .

Se  $k \neq \pi^4[q - 1]$ , então ...



# Recorrência

$\pi[q] :=$  maior comprimento de um **prefixo próprio**  
de  $P[1..q]$  que é **sufixo** de  $P[1..q]$   
 $:= \max\{k : k < q \text{ e } P[1..k] \text{ é sufixo de } P[1..q]\}$

$q$	0	1	2	3	4	5	6	7	8	9	10	11
$P$	'?'	a	b	a	b	b	a	b	a	b	b	a
$\pi$	-1	0	0	1	2	0	1	2	3	4	5	6

$$\pi[0] = -1$$

$$\pi[1] = 0$$

$$\pi[q] = \max \{ \pi^j[q-1] + 1 : P[\pi^j[q-1] + 1] = P[q] \}$$

Suponha  $P[0] = \text{"coringa"}$ .

# Compute-Prefix-Function (grosseiro)

COMPUTE-PREFIX-FUNCTION ( $P, m$ )

```
0   $\pi[1] \leftarrow 0$ 
1   $q \leftarrow 2$ 
2  enquanto  $q \leq m$  faça
3       $k \leftarrow q - 1$ 
4      repita
5           $i \leftarrow k$ 
6           $j \leftarrow q$ 
7          enquanto  $i > 0$  e  $P[i] = P[j]$  faça
8               $i \leftarrow i - 1$      $j \leftarrow j - 1$ 
9              se  $i > 0$ 
10                 então  $k \leftarrow k - 1$ 
11         até que  $i = 0$ 
12      $\pi[q] \leftarrow k$ 
13 devolva  $\pi$ 
```

# Compute-Prefix-Function (melhorado)

COMPUTE-PREFIX-FUNCTION ( $P, m$ )

```
0   $\pi[1] \leftarrow 0$ 
1   $q \leftarrow 2$ 
2  enquanto  $q \leq m$  faça
3       $k \leftarrow \pi[q - 1]$ 
4      repita
5           $i \leftarrow k + 1$ 
6           $j \leftarrow q$ 
7          enquanto  $i > 0$  e  $P[i + 1] = P[j]$  faça
8               $i \leftarrow i - 1$      $j \leftarrow j - 1$ 
9              se  $i > 0$ 
10                 então  $k \leftarrow \pi[k]$ 
11         até que  $i = 0$ 
12      $\pi[q] \leftarrow k + 1$ 
13 devolva  $\pi$ 
```

# Compute-Prefix-Function (melhor ainda)

COMPUTE-PREFIX-FUNCTION ( $P, m$ )

```
0   $\pi[1] \leftarrow 0$ 
1   $q \leftarrow 2$ 
2  enquanto  $q \leq m$  faça
3       $k \leftarrow \pi[q - 1]$ 
4      repita
5           $i \leftarrow k + 1$ 
6           $j \leftarrow q$ 
7          se  $P[i] = P[j]$ 
8              então  $i \leftarrow 0$ 
10         senão  $k \leftarrow \pi[k]$ 
11     até que  $i = 0$ 
12      $\pi[q] \leftarrow k + 1$ 
13 devolva  $\pi$ 
```

# Compute-Prefix-Function (limpo)

COMPUTE-PREFIX-FUNCTION  $(P, m)$

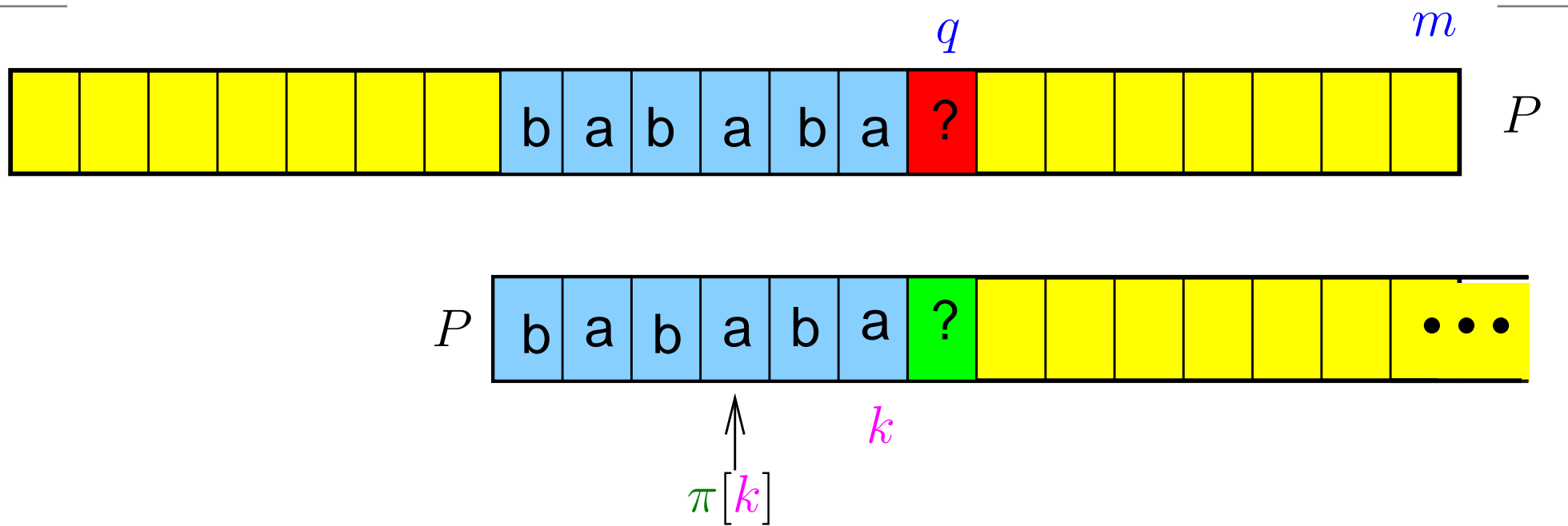
```
0   $\pi[1] \leftarrow 0$ 
1   $q \leftarrow 2$      $k \leftarrow 0$ 
2  enquanto  $q \leq m$  faça
3      enquanto  $k > 0$  e  $P[k + 1] \neq P[q]$  faça
4           $k \leftarrow \pi[k]$ 
5      se  $P[k + 1] = P[q]$ 
6          então  $k \leftarrow k + 1$ 
7       $\pi[q] \leftarrow k$ 
8  devolva  $\pi$ 
```

# Compute-Prefix-Function (limpo)

COMPUTE-PREFIX-FUNCTION ( $P, m$ )

```
0   $\pi[1] \leftarrow 0$ 
1   $q \leftarrow 2$    $k \leftarrow 0$ 
2  enquanto  $q \leq m$  faça
3      se  $P[k+1] = P[q]$  então  $\triangleright$  caso 1
4           $\pi[q] \leftarrow k + 1$ 
5           $q \leftarrow q + 1$ 
6           $k \leftarrow k + 1$ 
7      senão se  $P[k+1] \neq P[q]$  e  $k > 0$  então  $\triangleright$  caso 2
8           $k \leftarrow \pi[k]$ 
9      senão se  $P[k+1] \neq P[q]$  e  $k = 0$  então  $\triangleright$  caso 3
10          $\pi[q] \leftarrow 0$ 
11          $q \leftarrow q + 1$ 
12  devolva  $\pi$ 
```

# Correção



Relações invariantes: na linha 2 vale que

(i0)  $P[1..k]$  é **sufixo** de  $P[1..q-1]$

(i1)  $P[1..j]$  **não** é sufixo de  $P[1..q]$ , para  $j > k + 1$ .

# Consumo de tempo

Cada linha do algoritmo consome tempo  $\Theta(1)$ , exceto a linha 12 que pode consumir tempo  $O(m)$ .

Caso 1 e caso 3 ocorrem  $m$  vezes (total).

O número de ocorrências do caso 2 é  $\leq$  número de ocorrências do caso 1, pois

- $k$  nunca é negativo;
- $k$  é incrementado no caso 1; e
- $k$  é decrementado no caso 2.

Logo, o número total de iterações é  $\leq m + m = 2m$ .

O consumo de tempo do algoritmo é  $\Theta(m)$ .



# Conclusões

O consumo de tempo do algoritmo  
**COMPUTE-PREFIX-FUNCTION** é  $\Theta(m)$ .

O consumo de tempo do algoritmo **KMP-MATCHER**  
é  $\Theta(n + m)$ .

# Conjuntos disjuntos dinâmicos

CLR 22    CLRS 21

# Conjuntos disjuntos

Seja  $\mathcal{S} = \{S_1, S_2, \dots, S_n\}$  uma coleção de conjuntos disjuntos, ou seja,

$$S_i \cap S_j = \emptyset$$

para todo  $i \neq j$ .

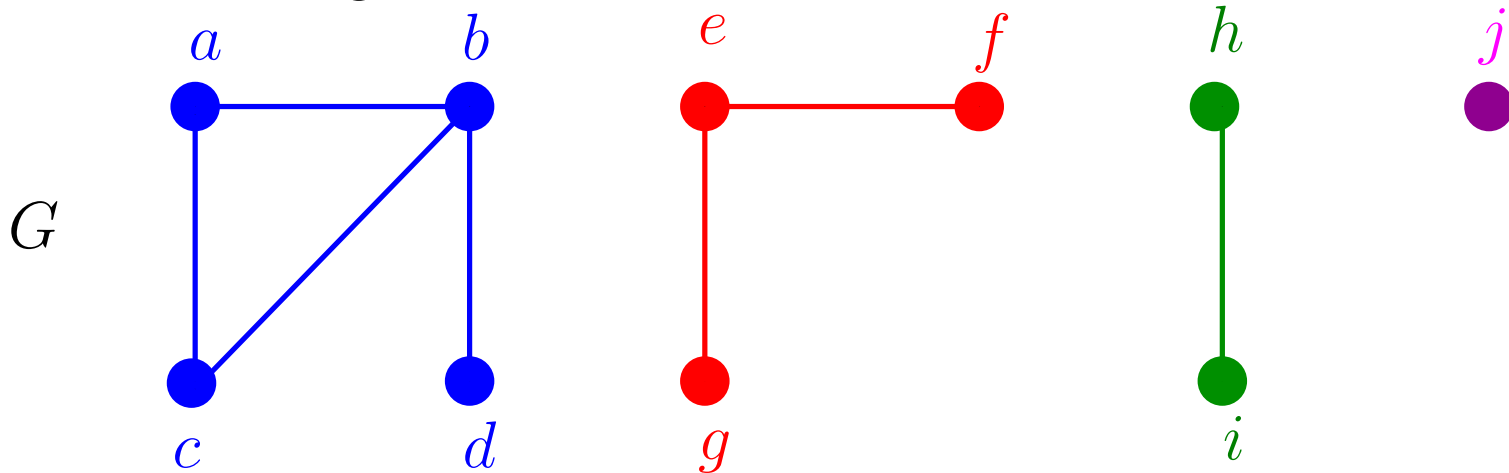
# Conjuntos disjuntos

Seja  $\mathcal{S} = \{S_1, S_2, \dots, S_n\}$  uma coleção de conjuntos disjuntos, ou seja,

$$S_i \cap S_j = \emptyset$$

para todo  $i \neq j$ .

Exemplo de coleção disjunta de conjuntos: **componentes conexos** de um grafo



componentes = conjuntos disjuntos de vértices

---

$$\{a, b, c, d\} \quad \{e, f, g\} \quad \{h, i\} \quad \{j\}$$

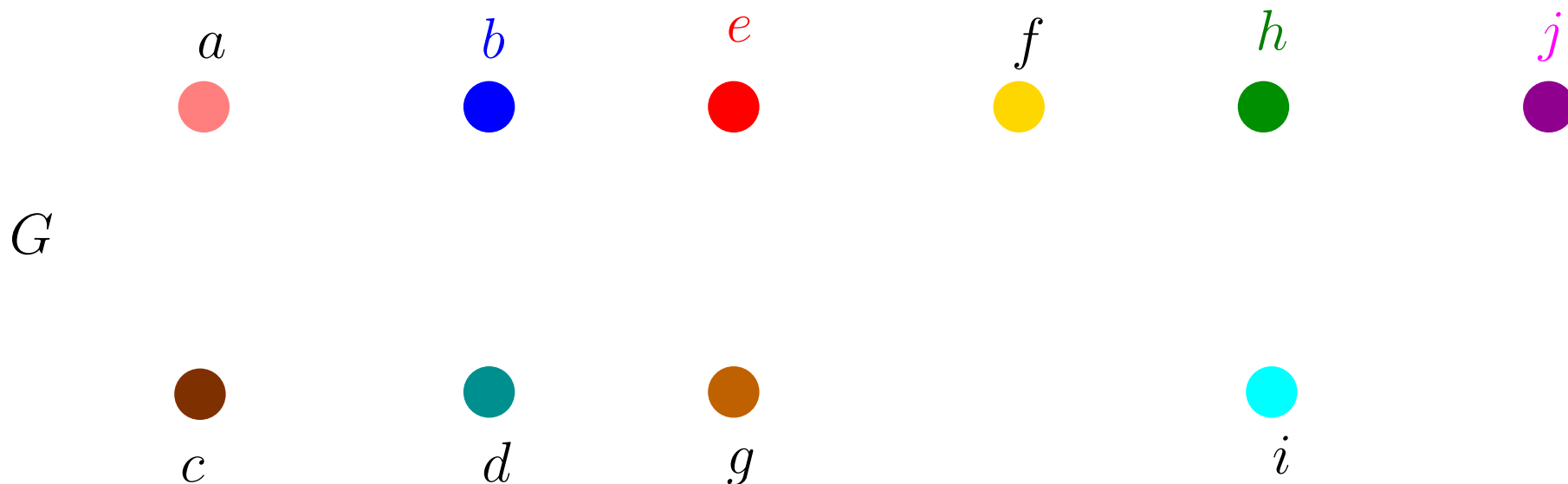
# Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

# Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



aresta

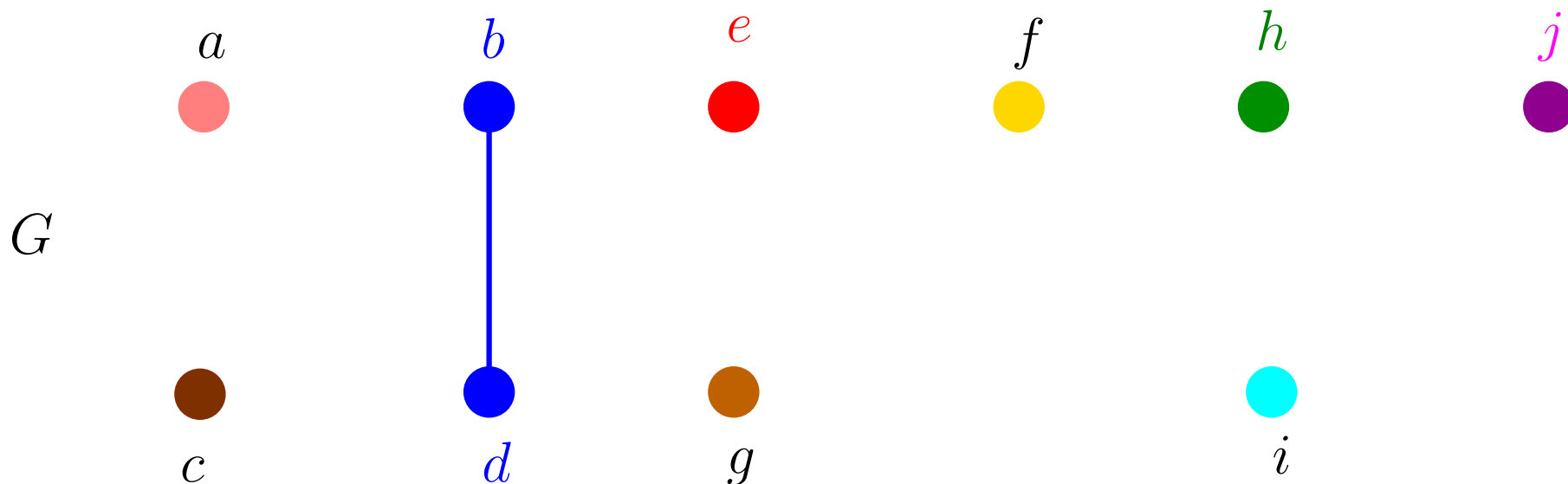
componentes

$\{a\}$   $\{b\}$   $\{c\}$   $\{d\}$   $\{e\}$   $\{f\}$   $\{g\}$   $\{h\}$   $\{i\}$   $\{j\}$

# Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



aresta

componentes

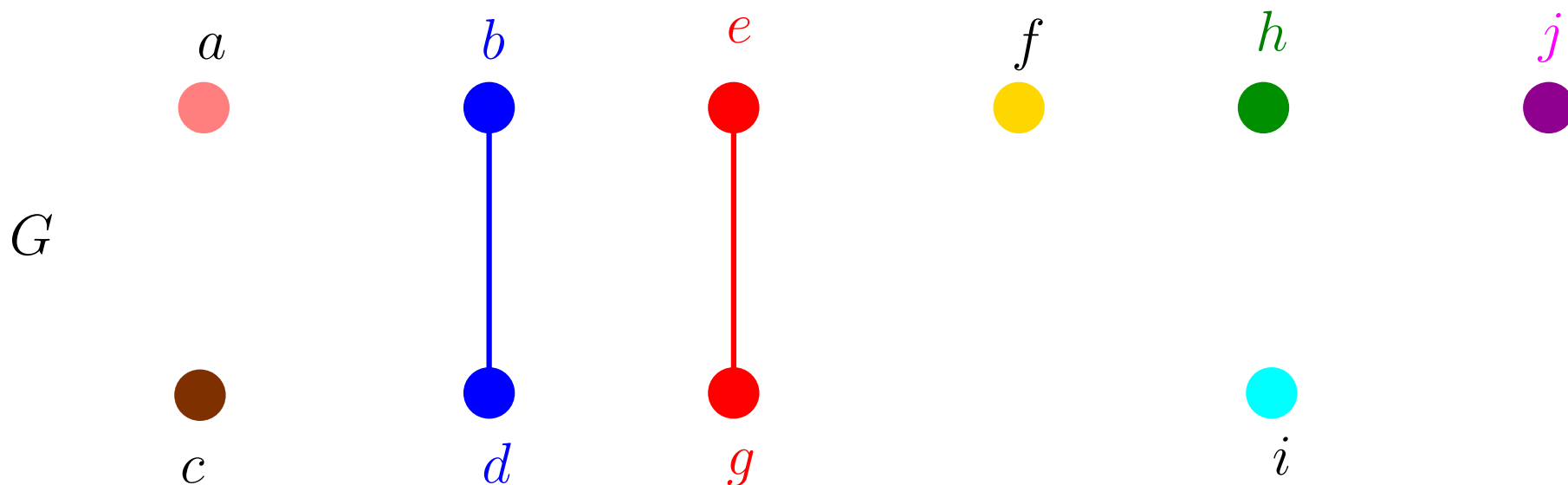
$(b, d)$

$\{a\}$   $\{b, d\}$   $\{c\}$   $\{e\}$   $\{f\}$   $\{g\}$   $\{h\}$   $\{i\}$   $\{j\}$

# Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



aresta

componentes

$(e, g)$

$\{a\}$

$\{b, d\}$

$\{c\}$

$\{e, g\}$

$\{f\}$

$\{h\}$

$\{i\}$

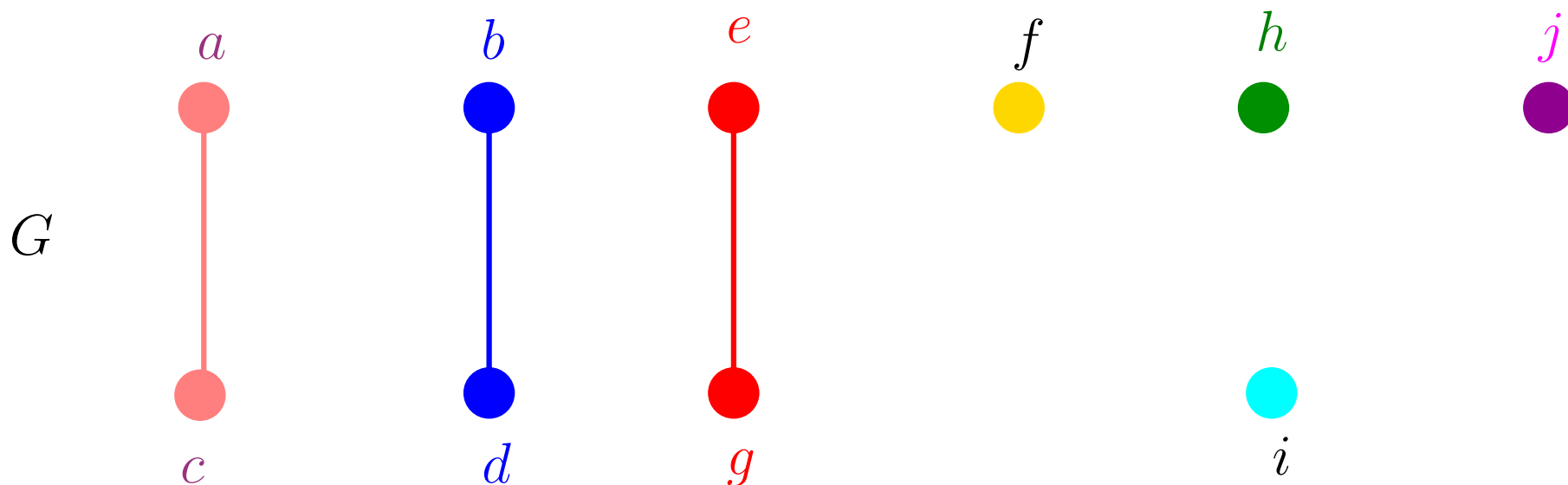
$\{j\}$



# Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



aresta

componentes

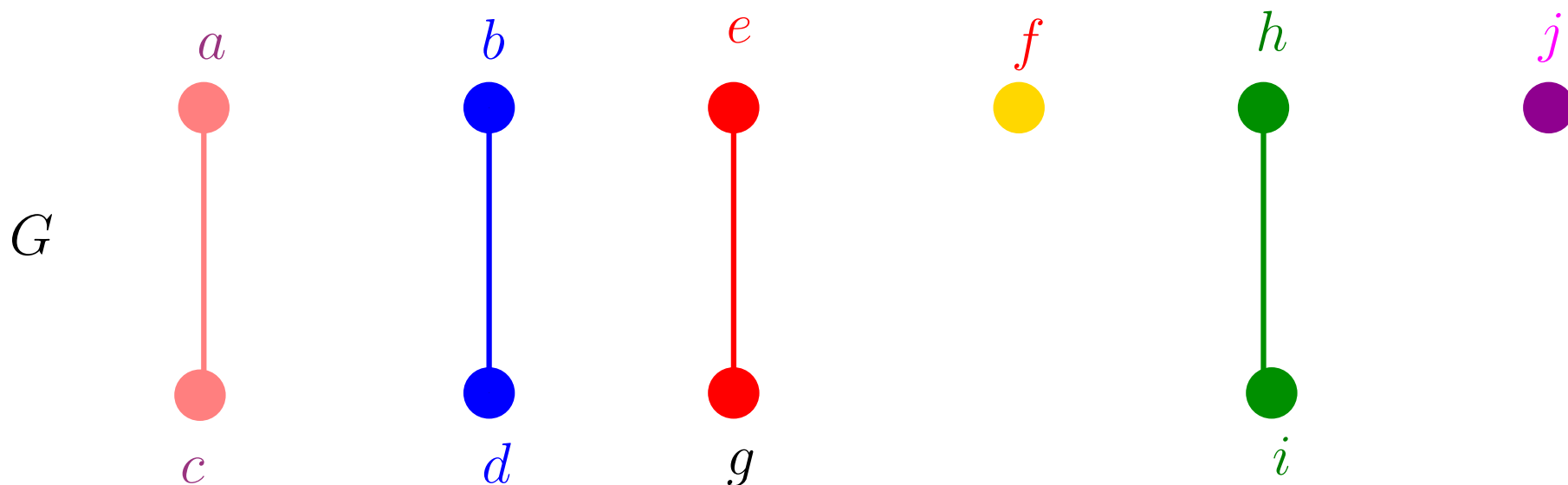
$(a, c)$

$\{a, c\}$   $\{b, d\}$   $\{e, g\}$   $\{f\}$   $\{h\}$   $\{i\}$   $\{j\}$

# Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



aresta

componentes

$(h, i)$

$\{a, c\}$

$\{b, d\}$

$\{e, g\}$

$\{f\}$

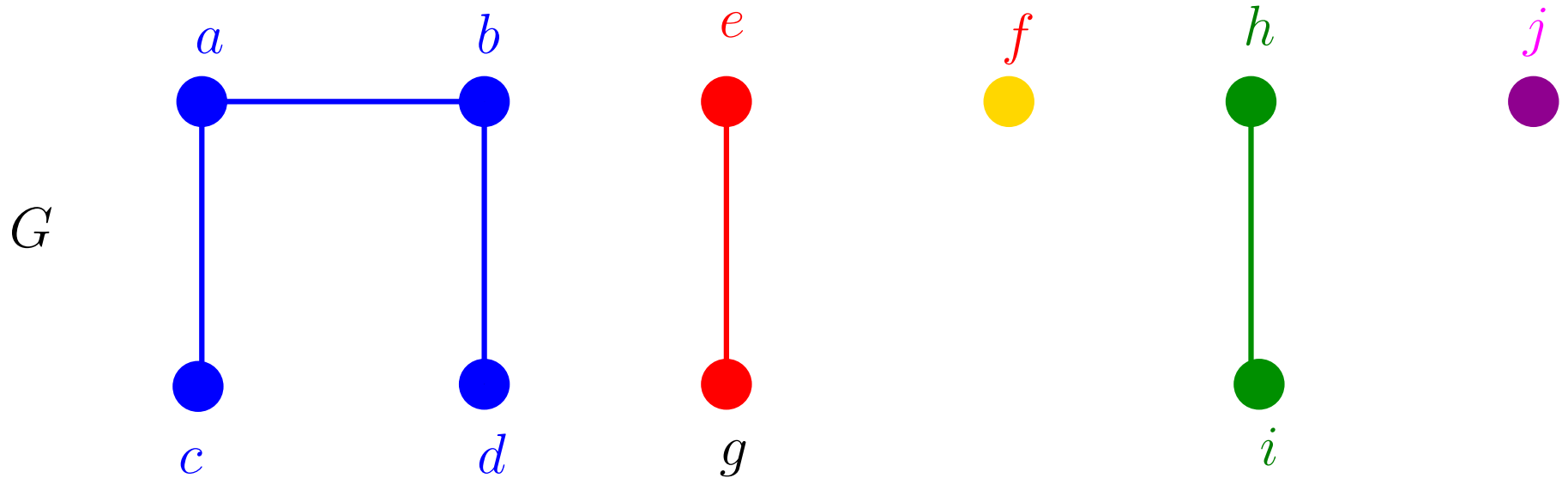
$\{h, i\}$

$\{j\}$

# Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



aresta

componentes

$(a, b)$

$\{a, b, c, d\}$

$\{e, g\}$

$\{f\}$

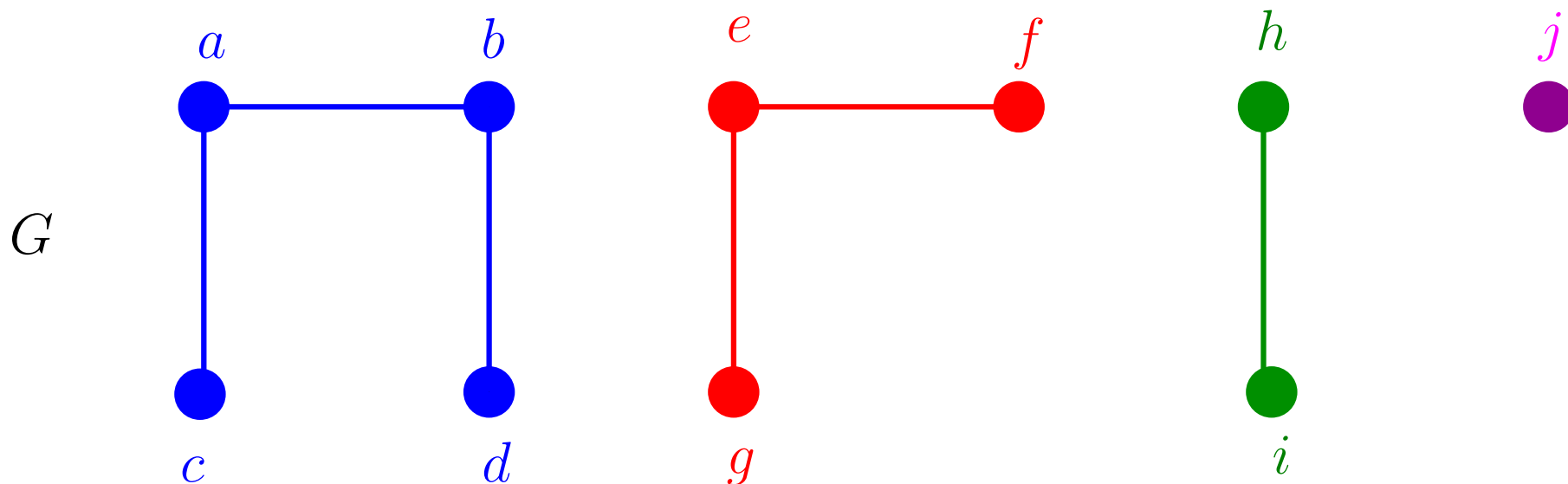
$\{h, i\}$

$\{j\}$

# Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



aresta

componentes

$(e, f)$

$\{a, b, c, d\}$

$\{e, f, g\}$

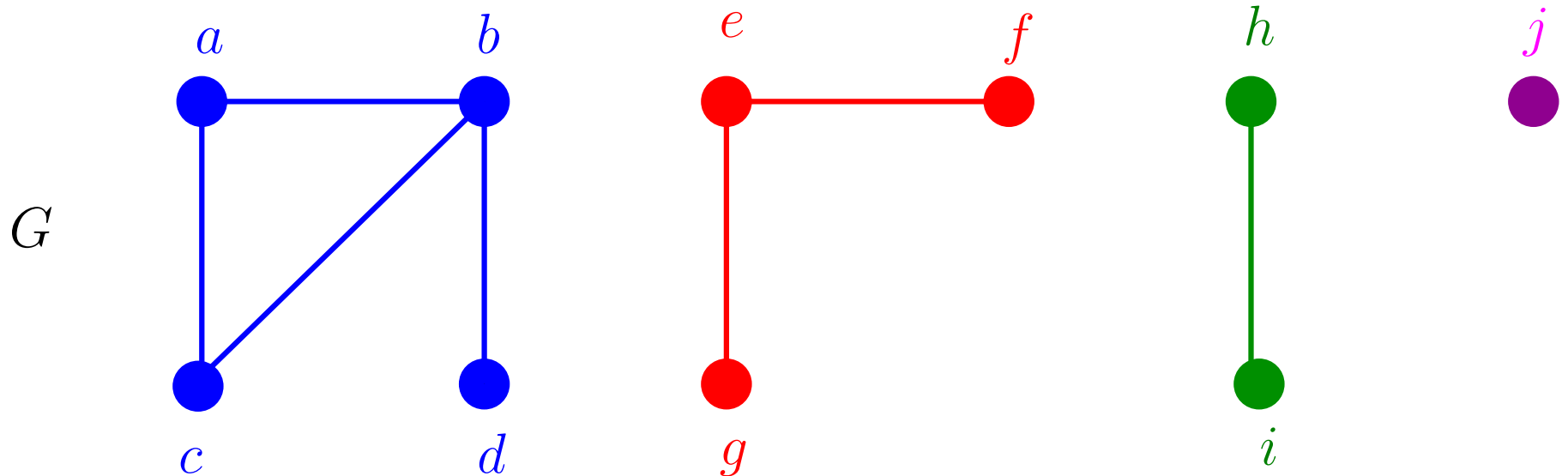
$\{h, i\}$

$\{j\}$

# Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



aresta

componentes

$(b, c)$

$\{a, b, c, d\}$

$\{e, f, g\}$

$\{h, i\}$

$\{j\}$

# Operações básicas

$\mathcal{S}$  coleção de conjuntos disjuntos.

Cada conjunto tem um representante.

**MAKESET** ( $x$ ):  $x$  é elemento novo

$$\mathcal{S} \leftarrow \mathcal{S} \cup \{x\}$$

**UNION** ( $x, y$ ):  $x$  e  $y$  em conjuntos diferentes

$$\mathcal{S} \leftarrow \mathcal{S} - \{S_x, S_y\} \cup \{S_x \cup S_y\}$$

$x$  está em  $S_x$  e  $y$  está em  $S_y$

**FINDSET** ( $x$ ): devolve representante do conjunto que contém  $x$

# Connected-Components

Recebe um grafo  $G$  e contrói uma representação dos componentes conexos.

## CONNECTED-COMPONENTS ( $G$ )

```
1  para cada vértice  $v$  de  $G$  faça
2      MAKESET ( $v$ )
3  para cada aresta  $(u, v)$  de  $G$  faça
4      se FINDSET ( $u$ )  $\neq$  FINDSET ( $v$ )
5          então UNION ( $u, v$ )
```

# Consumo de tempo

$n$  := número de vértices do grafo

$m$  := número de arestas do grafo

linha    consumo de **todas** as execuções da linha

---

$$1 \quad = \Theta(n)$$

$$2 \quad = n \times \text{consumo de tempo MAKESET}$$

$$3 \quad = \Theta(m)$$

$$4 \quad = 2m \times \text{consumo de tempo FINDSET}$$

$$5 \quad \leq n \times \text{consumo de tempo UNION}$$

---

$$\begin{aligned} \text{total} \quad &\leq \Theta(n + m) + n \times \text{consumo de tempo MAKESET} \\ &\quad + 2m \times \text{consumo de tempo FINDSET} \\ &\quad + n \times \text{consumo de tempo UNION} \end{aligned}$$



# Same-Component

Decide se  $u$  e  $v$  estão no mesmo componente:

**SAME-COMPONENT** ( $u, v$ )

- 1    **se** **FINDSET** ( $u$ ) = **FINDSET** ( $v$ )
- 2        **então devolva** **SIM**
- 3        **senão devolva** **NÃO**

# Algoritmo de Kruskal

Encontra uma **árvore geradora mínima** (CLRS 23).

**MST-KRUSKAL** ( $G, w$ )  $\triangleright G$  conexo

```
1   $A \leftarrow \emptyset$ 
2  para cada vértice  $v$  faça
3      MAKESET ( $v$ )
4  coloque arestas em ordem crescente de  $w$ 
5  para cada aresta  $uv$  em ordem crescente de  $w$  faça
6      se FINDSET ( $u$ )  $\neq$  FINDSET ( $v$ )
7          então  $A \leftarrow A \cup \{uv\}$ 
8              UNION ( $u, v$ )
9  devolva  $A$ 
```

“Avô” de todos os algoritmos gulosos.

# Conjuntos disjuntos dinâmicos

Seqüência de operações MAKESET, UNION, FINDSET

M M M U F U U F U F F F U F



$n$

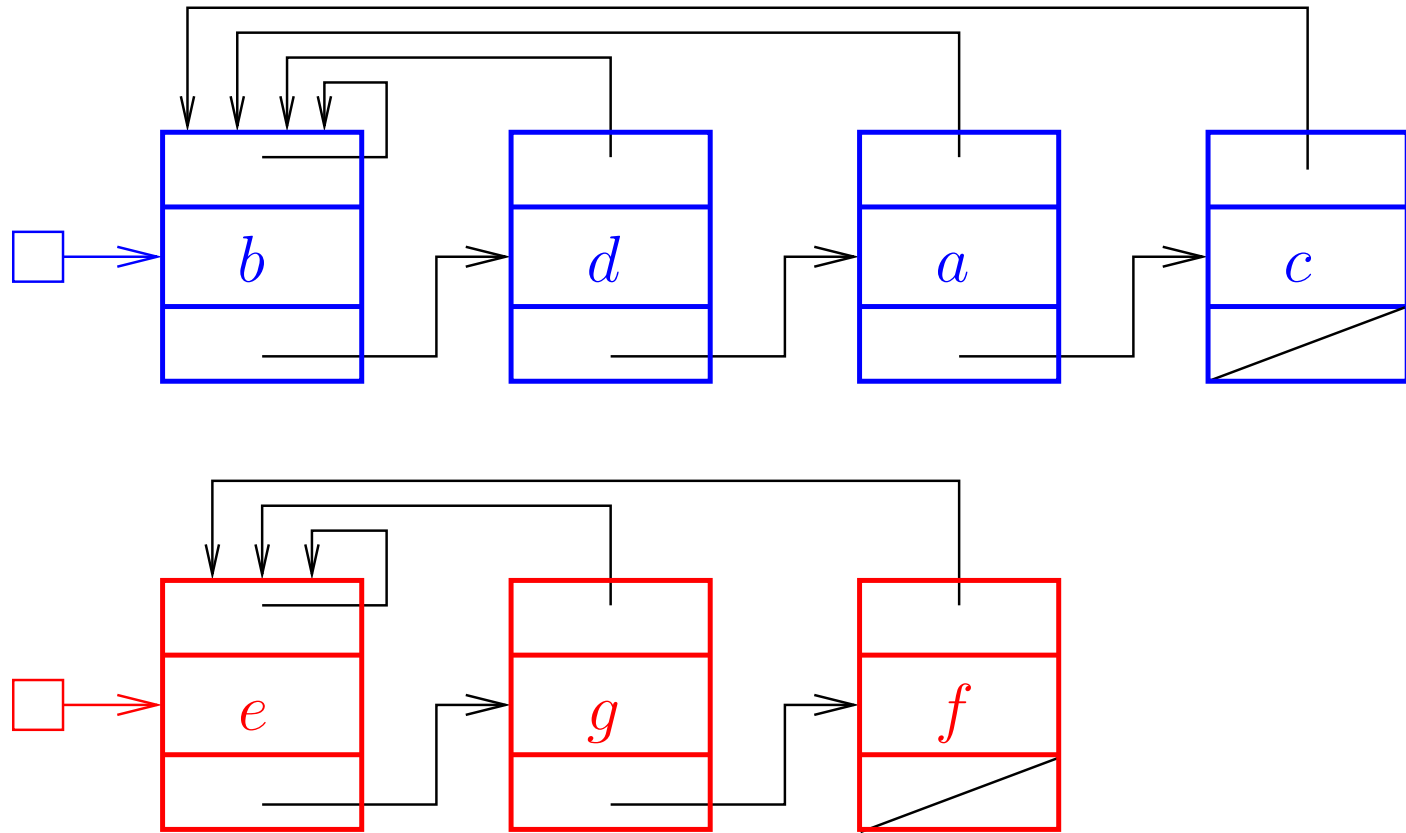


$m$

Que estrutura de dados usar?

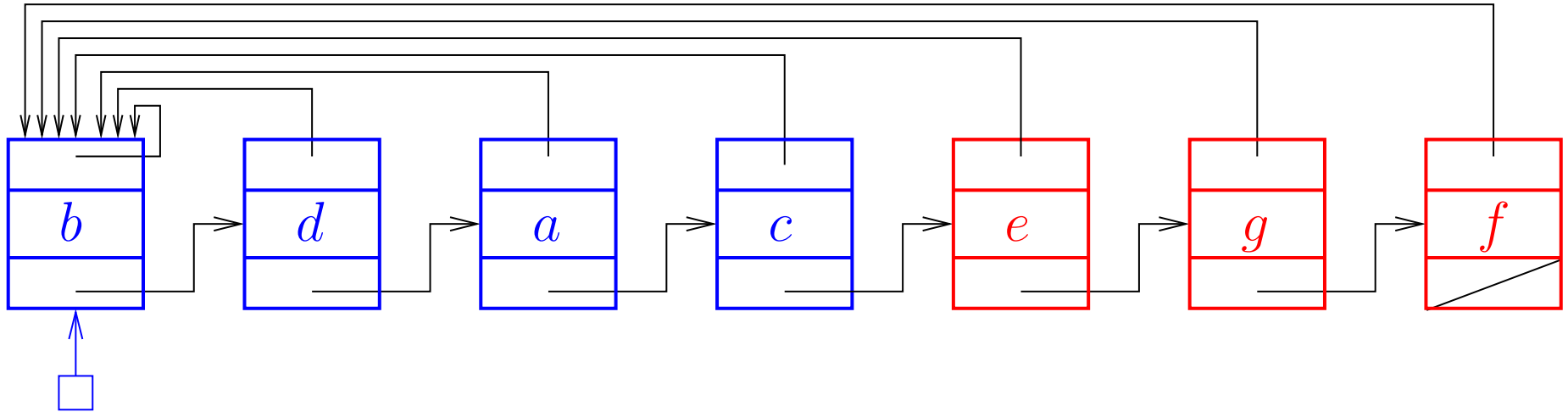
Compromissos (*trade-offs*)

# Estrutura de listas ligadas



- cada conjunto tem um representante (início da lista)
- cada nó  $x$  tem um campo  $repr$
- $repr[x]$  é o representante do conjunto que contém  $x$

# Estrutura de listas ligadas



**UNION** (*a*, *e*) : atualiza apontador para o representante.

# Consumo de tempo

Operação	número de objetos atualizados
MAKESET( $x_1$ )	1
MAKESET( $x_2$ )	1
⋮	1
MAKESET( $x_n$ )	1
UNION( $x_1, x_2$ )	1
UNION( $x_2, x_3$ )	2
⋮	⋮
UNION( $x_{n-1}, x_n$ )	$n - 1$
<hr/>	
total	$= \Theta(n^2) = \Theta(m^2)$

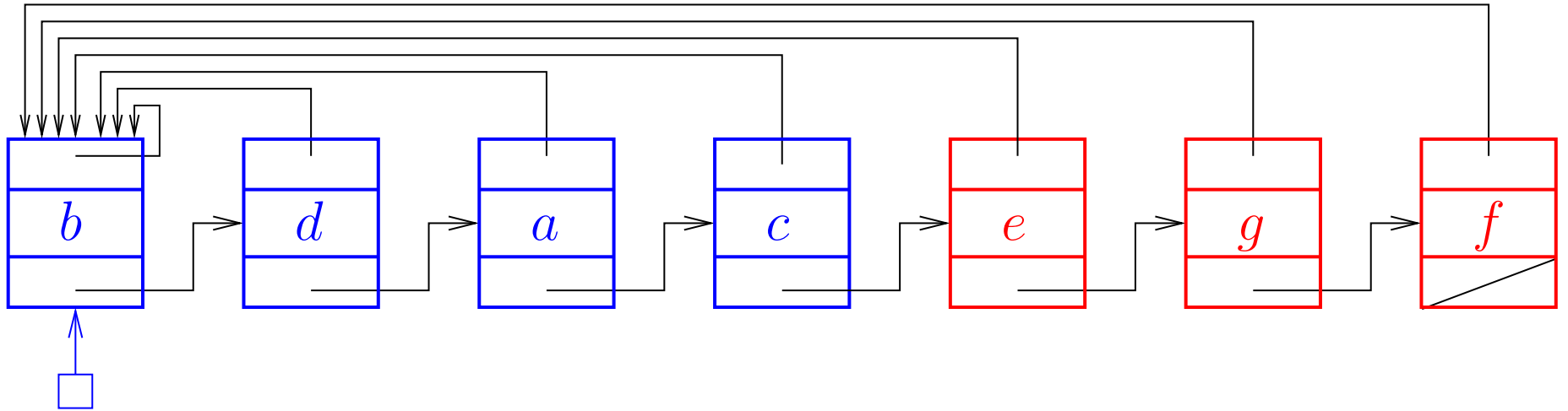
# Consumo de tempo

MAKESET	$\Theta(1)$
UNION	$O(n)$
FINDSET	$\Theta(1)$

Uma seqüência de  $m$  operações pode consumir tempo  $\Theta(m^2)$  no pior caso.

Consumo de tempo amortizado de cada operação é  $O(m)$ .

# Melhoramento: *weighted-union*



- cada representante armazena o comprimento da lista
- a lista menor é concatenada com a maior

Cada objeto  $x$  é atualizado  $\leq \lg n$ :

cada vez que  $x$  é atualizado o tamanho da lista dobra.



# Conclusão

Se conjuntos disjuntos são representados através de *listas ligadas* e *weighted-union* é utilizada, então uma seqüência de  $m$  operações MAKESET, UNION e FINDSET, sendo que  $n$  são MAKESET, consome tempo  $O(m + n \lg n)$ .